Knowledge Compilation and Weighted Model Counting for Inference in Probabilistic Logic Programs

Jonas Vlasselaer and **Angelika Kimmig** and **Anton Dries** and **Wannes Meert** and **Luc De Raedt**
Department of Computer Science, KU Leuven, Belgium
{firstname.lastname}@cs.kuleuven.be

Abstract

Over the last decade, building on advances in the areas of knowledge compilation and weighted model counting has drastically increased the scalability of inference in probabilistic logic programs. In this paper, we provide an overview of how this has been possible and point out some open challenges.

1 Introduction

Many real world reasoning tasks, such as gene interaction networks, social networks and web-page classification, involve both relational structure and uncertainty. This caused a significant interest in statistical relational learning (Getoor and Taskar 2007; De Raedt et al. 2008), probabilistic programming (Pfeffer 2014; De Raedt and Kimmig 2015; Goodman and Stuhlmüller 2014) and probabilistic databases (Suciu et al. 2011), which all address this combination. Probabilistic logic programming (PLP) languages such as PRISM (Sato and Kameya 2001), ICL (Poole 1993), ProbLog (De Raedt, Kimmig, and Toivonen 2007), LPADs (Vennekens, Verbaeten, and Bruynooghe 2004) and CP-logic (Vennekens, Denecker, and Bruynooghe 2009) form one stream of work in these fields. These formalisms extend the logic programming language Prolog with *probabilistic* choices on which facts are `true` or `false`.

One key inference task in PLP is to compute the probability that a given ground atom (the *query*) holds in a probabilistic logic program, possibly given some evidence on the truth values of other atoms. This inference task can be reduced to the well-studied task of weighted model counting (WMC) (Chavira and Darwiche 2008), for which various state-of-the-art solvers are available. Within recent PLP systems, WMC is often supported by knowledge compilation.

In this paper, we provide an overview of such approaches, with a special focus on how knowledge compilation and weighted model counting have helped to drastically increase scalability of PLP inference over the last decade. Furthermore, we also discuss the challenges that are still open. Specifically, Section 2 provides background on PLP inference and its reduction to weighted model counting on a propositional formula. Sections 3 and 4 discuss the most common approaches to construct this propositional formula and to compute its weighted model count, respectively. Section 5 presents techniques that integrate construction and model counting, and Section 6 concludes with a discussion of open problems.

2 Inference in Probabilistic Logic Programs

Many probabilistic programming languages, including PRISM (Sato and Kameya 2001), ICL (Poole 1993), ProbLog (De Raedt, Kimmig, and Toivonen 2007), and LPADs (Vennekens, Verbaeten, and Bruynooghe 2004) are based on Sato's distribution semantics (Sato 1995). In this paper, we use ProbLog as it has the simplest syntax of these languages; for a general overview of PLP and more details on the relation between these languages, we refer to (De Raedt and Kimmig 2015).

$0.4 :: \texttt{edge}(b, a).$ $\quad 0.3 :: \texttt{edge}(b, c).$
$0.8 :: \texttt{edge}(a, c).$
$\texttt{path}(X, Y)$:- $\texttt{edge}(X, Y).$
$\texttt{path}(X, Y)$:- $\texttt{edge}(X, Z), \texttt{path}(Z, Y).$

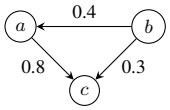

Figure 1: A probabilistic logic program modeling a graph.

Probablistic Logic Programs

A ProbLog program $\mathcal{P}$ consists of a set of probabilistic facts $\mathcal{F}$ and a logic program, i.e., a set of rules $\mathcal{R}$. A *rule* is a universally quantified expression of the form $\texttt{h} :- \texttt{b}_1, ..., \texttt{b}_\texttt{n}$ where `h` is an atom and the $\texttt{b}_\texttt{i}$ are literals. The atom `h` is called the *head* of the rule and $\texttt{b}_1, ..., \texttt{b}_\texttt{n}$ the *body*, representing the conjunction $\texttt{b}_1 \wedge ... \wedge \texttt{b}_\texttt{n}$. Intuitively, such a rule states that whenever the body is true, the head has to be true as well.[1] A *probabilistic fact* is a rule annotated with a probability `p` that has *true* as its body. It is written more compactly as $\texttt{p} :: \texttt{f}$. As common, we assume that none of the probabilistic facts unify with the head of a rule.

An example is depicted in Figure 1. This program contains the representation of a graph (as a set of probabilistic edges), and the definition of a path between two nodes X

[1] We refer to (Nilsson and Maluszynski 1995) for more details on logic programming in general.

and Y. A path either consists of a direct edge between X and Y, or of an edge between X and a third node Z and a path from Z to Y. The lower case letters (a, b and c) represent constants, and the upper case letters (X, Y and Z) represent logical variables.

Inference

A ProbLog program specifies a probability distribution over its *Herbrand interpretations*, also called possible worlds. Every probabilistic fact[2] `p :: f` independently takes a value `true` (with probability p) or `false` (with probability $1-p$). A *total choice* $C \subseteq \mathcal{F}$ assigns a truth value to every probabilistic fact. Then, logical deduction on the corresponding logic program $C \cup \mathcal{R}$ results in a model of the theory. The probability of this model is that of C. Possible worlds that do not correspond to any total choice have probability zero.

The task we focus on in this paper is to compute the probability of a single query q, which is defined as the sum over all total choices whose program entails q:

$$\Pr(q) := \sum_{C \subseteq \mathcal{F} : C \cup \mathcal{R} \models q} \prod_{f_i \in C} p_i \cdot \prod_{f_i \in \mathcal{F} \setminus C} (1 - p_i) \,. \qquad (1)$$

Whereas this formula defines the semantics of ProbLog, it explicitly iterates over all total choices that entail the query. This approach is infeasible in practice.

Weighted Model Counting

The task of probabilistic inference in PLP (cf. Equation 1) is in direct correspondence with that of weighted model counting. We can exploit this to tackle the problem in two steps: (1) construct a propositional formula representing the possible worlds, and (2) perform weighted model counting on this formula. More specifically, ground probabilistic facts f_i correspond to propositional random variables, probabilities provide weights, and the sum is over a propositional formula representing all possible worlds where the query is true, i.e.,

$$WMC(\lambda) := \sum_{I \subseteq V : I \models \lambda} \prod_{a \in I} w(a) \cdot \prod_{a \in V \setminus I} w(\neg a) \,. \qquad (2)$$

where λ is a formula over a set of propositional variables V, the weight function $w(\cdot)$ assigns a real number to every literal for an atom in V, and I is the set of interpretations of V, i.e. the set of all possible truth value assignments to variables in V.

For the weight function we have that $w(f_i) = p_i$ and $w(\neg f_i) = 1 - p_i$ for probabilistic facts $p_i :: f_i$, and $w(a) = w(\neg a) = 1$ else. Once we have a formula λ such that for every total choice $C \subseteq \mathcal{F}$, $C \wedge \lambda \models q \leftrightarrow C \cup \mathcal{R} \models q$, we can compute $\Pr(q)$ as WMC(λ). While λ may use variables besides the probabilistic facts, their values have to be uniquely defined for each total choice.

We note that the reduction also allows us to perform approximate inference with lower and upper bounds. Specifically, we have that whenever $\lambda_l \models \lambda \models \lambda_u$, then $WMC(\lambda_l) \leq WMC(\lambda) \leq WMC(\lambda_u)$. This is especially useful if constructing the full formula λ is infeasible.

[2]For ease of notation, we assume that $\mathcal{F}$ is ground.

Knowledge Compilation

The conversion of a probabilistic logic program towards a propositional formula is only effective in case computation of the weighted model count can be done efficiently. In most PLP systems, weighted model counting is supported by knowledge compilation. The idea here is to compile the formula into a more tractable target representation after which inference can be performed in time linear in the size of the compiled structure.

The knowledge compilation languages of interest in this paper are disjunctive normal form (`DNF`), conjunctive normal form (`CNF`), models (`MODS`), deterministic decomposable negation normal form (`d-DNNF`), ordered binary decision diagram (`OBDD`) and sentential decision diagram (`SDD`) (Darwiche and Marquis 2002; Darwiche 2011). Within the knowledge compilation map, `MODS`, a `DNF` where each conjunction contains all variables, is the most tractable language. More useful target languages are `d-DNNF`, `OBDD` and `SDD`. While `d-DNNF` is the most general of these languages and comes with the least restrictions, the advantage of `OBDD` and `SDD` is that they allow for incremental formula construction.

3 Constructing the Propositional Formula

It is straightforward to convert a probabilistic logic program to a propositional formula in `MODS` representation by simply enumerating all total choices (conjunctions of literals) that entail the query. In the example of Figure 1, the formula for query `path`(b, c), where we use xy for `edge`(x, y) being in the model, would be

$$\begin{array}{lll} (ba \wedge bc \wedge ac) & \vee & (\neg ba \wedge bc \wedge ac) \quad \vee \\ (ba \wedge bc \wedge \neg ac) & \vee & (\neg ba \wedge bc \wedge \neg ac) \quad \vee \\ (ba \wedge \neg bc \wedge ac) & & \end{array}$$

listing a total of 5 total choices (or subgraphs) explicitly. However, this is clearly infeasible for all but the tiniest programs, and existing PLP inference techniques therefore use other representations, most often `DNF` or `CNF`. We discuss these two alternatives next.

3.1 Explanation Based Conversion to `DNF`

The first approach to construct a more efficient representation of λ is based on the observation that, instead of listing *total* choices entailing the query, we can also list *partial* choices. A partial choice is a truth value assignment to a subset of the probabilistic facts where all total choices extending the assignment entails the query. We refer to such partial choices as *explanations* of the query. As each explanation corresponds to a set of total choices, a `DNF` of explanations is often more compact than a `MODS` representation. In our graph example, this means listing *paths* instead of *subgraphs*, i.e., for query `path`(b, c) we would get $bc \vee (ba \wedge ac)$.

In probabilistic logic programming, a covering set of explanations can easily be obtained by backtracking over all successful derivations of the query, i.e., using standard logic programming inference such as SLD-resolution. Figure 2 shows the SLD-tree obtained by evaluating the query `path`(b, c) on the example from Figure 1. Each success branch corresponds to one explanation in the `DNF`.

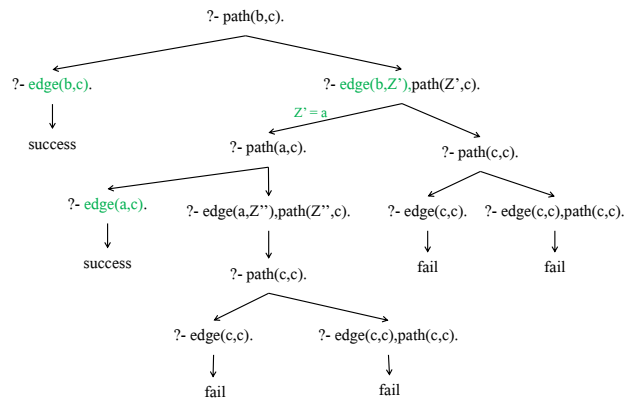

Figure 2: The SLD-tree of our example and the query $\texttt{path}(b, c)$. We find two explanations.

The idea of a DNF based on explanations has been used, for instance, in the context of PHA, ICL, PRISM, ProbLog and LPADs (Poole 1993; 2000; Sato and Kameya 2001; De Raedt, Kimmig, and Toivonen 2007; Riguzzi 2007; Riguzzi and Swift 2011), cf. also Sections 4.1 and 4.3.

Approximate Inference

It is also common to approximate inference by selecting a subset of explanations only, which then results in a lower bound after model counting. Possible selection criteria include all explanations up to a certain length, all explanations for which the probability of the partial choice is above a certain threshold, or the k explanations (for a given number k) for which this probability is the highest. In our example, the probabilities of the explanations are 0.3 (for bc) and $0.8 \cdot 0.4 = 0.32$ (for $ba \wedge ac$), and with $k = 1$ we would thus select the longer explanation.

3.2 Rule Based Conversion to CNF

The standard input format for most off-the-shelf weighted model counting solvers is a CNF. Hence, it is attractive to convert a probabilistic logic program into a CNF.

For programs without cyclic dependencies, such as our example, conversion to CNF requires three steps: (1) find all ground rules used in some proof of the query, (2) apply Clark's completion (Nilsson and Maluszynski 1995) to obtain a formula in propositional logic, and (3) rewrite this formula to CNF. In our example, the relevant ground rules for query $\texttt{path}(b, c)$ are

$$\begin{aligned}\texttt{path}(b,c) &\texttt{ :- } \texttt{edge}(b,c).\\ \texttt{path}(b,c) &\texttt{ :- } \texttt{edge}(b,a), \texttt{path}(a,c).\\ \texttt{path}(a,c) &\texttt{ :- } \texttt{edge}(a,c).\end{aligned}$$

The propositional formula contains one subformula for each ground atom appearing in the head of some rule. This subformula states that the propositional variable corresponding to this atom is equivalent to the disjunction of all its rules bodies, i.e., we get

$$\begin{aligned}& p_{pc} \leftrightarrow bc \vee (ba \wedge p_{ac})\\ \wedge\ & p_{ac} \leftrightarrow ac\end{aligned}$$

We omit the CNF.

For programs with cyclic dependencies, however, Clark's completion does not correctly capture the semantics. Consider our example, but with an extra probabilistic fact $\texttt{edge}(a, b)$ which introduces a directed cycle in the graph. Now, the ground program for $\texttt{path}(b, c)$ is

$$\begin{aligned}\texttt{path}(b,c) &\texttt{ :- } \texttt{edge}(b,c).\\ \texttt{path}(b,c) &\texttt{ :- } \texttt{edge}(b,a), \texttt{path}(a,c).\\ \texttt{path}(a,c) &\texttt{ :- } \texttt{edge}(a,c).\\ \texttt{path}(a,c) &\texttt{ :- } \texttt{edge}(a,b), \texttt{path}(b,c).\end{aligned}$$

The completion of this set of rules can be satisfied by setting $\texttt{path}(a, c), \texttt{path}(b, c), \texttt{edge}(b, a), \texttt{edge}(a, b)$ to `true` and everything else to `false`, but this is not a valid possible world under logic programming semantics.

It is well-known that this problem can be solved by rewriting the ground program, and state-of-the-art inference for ProbLog relies on this approach (Fierens et al. 2015). This rewriting step, however, introduces auxiliary atoms and may drastically increase the size of the ground program, and thus also of the CNF. Intuitively, rewriting can be seen as duplicating ground atoms, and using the different copies in different contexts. E.g., one could rewrite the above example to

$$\begin{aligned}\texttt{path}(b,c) &\texttt{ :- } \texttt{edge}(b,c).\\ \texttt{path}(b,c) &\texttt{ :- } \texttt{edge}(b,a), \texttt{aux\_path}(a,c).\\ \texttt{path}(a,c) &\texttt{ :- } \texttt{edge}(a,c).\\ \texttt{path}(a,c) &\texttt{ :- } \texttt{edge}(a,b), \texttt{aux\_path}(b,c).\\ \texttt{aux\_path}(b,c) &\texttt{ :- } \texttt{edge}(b,c).\\ \texttt{aux\_path}(a,c) &\texttt{ :- } \texttt{edge}(a,c).\end{aligned}$$

The size of the transformed program, and thus the CNF, increases dramatically with the number of cyclic dependencies and limits the scalability of inference based on CNF. For example, the conversion of the complete grounding for a path query on a fully connected undirected graph with only 10 nodes (and thus 90 probabilistic facts) results in a CNF formula with 26995 variables and 109899 clauses.

Approximate Inference

One way to approximate inference based on the CNF is provided by (Renkens et al. 2014). Here, the explanation search on the CNF is formulated as a weighted partial MaxSAT problem and is further encoded such that solutions can iteratively be obtained from a standard weighted MaxSAT solver. These explanations then provide a lower bound for the probability of the query. An upper bound can be obtained based on explanations for the negation of the query.

4 Weighted Model Counting

We now discuss various options to perform weighted model counting on the formulae constructed above, including compilation of a propositional formula into a suitable target representation. We first specifically consider DNF and CNF formulae, and then discuss general methods usable with any representation (including DNF and CNF).

4.1 DNF formula

If all explanations of the query are *mutually exclusive*, i.e., no two partial choices can be extended to the same total choice, the weighted model count or probability can directly be computed on the DNF. To benefit from this, PHA (Poole 1993) and PRISM (Sato and Kameya 2001) require that programs ensure mutually exclusive explanations. This excludes certain natural models such as our path example (note that the possible world where all edges are `true` contains both of the paths that prove our query).

An alternative is to explicitly *disjoin* the explanations in the DNF, for instance, using the inclusion-exclusion-principle, or by systematically adding additional choices to explanations. The former has been used for probabilistic Datalog (pD) (Fuhr 2000), where it is reported to scale to about ten explanations. An example of the latter has been proposed for ICL (Poole 2000), but again has limited scalability.

A much more scalable approach is to compile the DNF into a target language suitable for weighted model counting. Many of-the-shelf compilers, however, do not support DNF, or general formulae, as input language. We discuss compilation for general formulae below.

4.2 CNF Formula

A formula in CNF is the standard input format for most weighted model counting tools. Hence, conversion into CNF allows one to easily integrate and compare different solvers. One common approach is compilation to d−DNNF or SDD as done in ProbLog (Fierens et al. 2015; Vlasselaer et al. 2014). Alternatives are on-the-fly counters which compute the WMC without keeping a trace, i.e. without explicitly compiling the formula. For approximate counting, one could rely on sampling approaches, such as MC-SAT (Poon and Domingos 2006), which only return an estimate of the WMC.

4.3 Any Formula

ProbLog (De Raedt, Kimmig, and Toivonen 2007) was the first system that used knowledge compilation to solve the problem of overlap between explanations, with OBDD as the target compilation language. Exact inference effectively scaled up to, e.g., path queries with a few hundred thousand explanations. Instead of explicitly representing the set of explanations as DNF, a more compact internal representation is used that exposes repeated subformulae to the OBDD compiler. This approach has later been adopted also for LPADs in the cplint (Riguzzi 2007) and PITA (Riguzzi and Swift 2011) systems , where the latter directly represents the formula as OBDD during explanation search.

These approaches crucially rely on the fact that OBDD allows for efficient bottom-up compilation, i.e., any propositional formula can be used as input. Bottom-up compilation relies on an efficient implementation of the *apply-operator* which allows for Boolean operations on formulae. As a consequence, OBDD could be easily replaced by other target languages with this property, such as SDD.

5 Integrating Conversion and Counting

The approaches discussed so far completely separate logical reasoning (formula construction) from probabilistic reasoning (weighted model counting). For approximate inference, at most the probabilities are used to guide the selection of explanations, i.e. one searches for explanations with the highest probability. Avoiding this separation has been shown beneficial in a number of settings, however.

First, when constructing a DNF with a fixed maximal number k of explanations for approximate inference, it makes sense to chose the explanations based on how much they contribute to the weighted model count of the final selection, rather than on their individual probability. This contribution can efficiently be evaluated on an OBDD representation of the current selections. With this approach, one can achieve results with optimality guarantees (Renkens, Van den Broeck, and Nijssen 2012).

Second, conversion into CNF is often prohibitively expensive as it requires explicit handling of cyclic dependencies. One way to avoid CNF conversion is proposed by (Vlasselaer et al. 2015). Instead of constructing a single formula, they iteratively construct a collection of formulae, one for each ground atom. The formulae are directly represented as SDDs. Formulae for probabilistic facts are initialized to their corresponding propositional variable and all other formulae to `false`. The approach then iteratively updates formulae based on rules. Using SDDs allows one to efficiently construct these formulae as well as to recognize when no further updates are needed (i.e., for all atoms, the previous and next formulae are logically equivalent). An additional advantage of this technique is that it avoids the need for a compact internal representation, as often used for the explanation based conversion.

An alternative way to avoid conversion to CNF is to modify a propositional model counter to count stable models of a ground logic program rather than models of a propositional formula. This has been done with the DSHARP knowledge compiler (Aziz et al. 2015).

6 Discussion and Challenges

By building on top of advanced model counting techniques, probabilistic logic programming systems have tackled problems like the disjoint-sum problem. Furthermore, scalability is significantly improved. Standardization across the field makes it possible to easily interchange different solvers and to use the best approach for a given problem. While interchangeability is partially thanks to the strict input formats (e.g., a full formula in CNF), this also poses a challenges to probabilistic logic inference techniques.

As shown throughout the text, the conversion of a logic program into a propositional formula is not straightforward. For example, generating a CNF is often infeasible for highly cyclic programs. As a consequence, the bottleneck for certain programs is situated even before calling weighted model counters or knowledge compilers. Several results have shown that it is more efficient to interweave logic programming reasoning and knowledge compilation (Renkens, Van den Broeck, and Nijssen 2012; Vlasselaer et al. 2015;

Aziz et al. 2015). That way, model counting can be used throughout the entire chain and we can build upon the already compiled formulae to keep them small.

The use of WMC during conversion has shown to be beneficial when compiling approximate formulae. Hence, a **first challenge** for the community is to provide knowledge compilers that, given a weighted formula and resource restrictions (e.g. time or memory), compile a formula that maximizes the WMC. To our knowledge, different tools for approximate weighted model counting exist but none of the knowledge compilers support approximate compilation.

Interleaving reasoning on the logic program with construction of a propositional formula for WMC heavily relies on efficient operations on already compiled formulae. Hence, a **second challenge** for the community is to provide more tools with an interactive interface that allows one to incrementally build and manipulate formulae. To our knowledge, only a subset of the current tools provide such an interface, e.g. compilers to `OBDD` or `SDD`, while most other solvers require as input a propositional formula in a strict format, e.g. `CNF` or `DNF`.

All techniques we discussed so far require a propositional input and, as such, the first step of PLP inference techniques involves grounding the program, either explicitly or during explanation search. A **third challenge** for the community is to provide tools that allow us to omit this grounding step by using first order knowledge compilation. While the first steps into this direction have already been taken (Van den Broeck, Meert, and Darwiche 2014), more work is certainly needed.

Acknowledgments

The authors wish to thank Guy Van den Broeck for valuable discussions. Jonas Vlasselaer is supported by IWT (agency for Innovation by Science and Technology). Angelika Kimmig is supported by FWO (Research Foundation-Flanders). This work is partially supported by FWO projects and GOA 13/010.

References

Aziz, R. A.; Chu, G.; Muise, C.; and Stuckey, P. J. 2015. Stable Model Counting and Its Application in Probabilistic Logic Programming. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*.

Chavira, M., and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artif. Intell.* 172(6-7):772–799.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of AI Research* 17:229–264.

Darwiche, A. 2011. SDD: A New Canonical Representation of Propositional Knowledge Bases. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*.

De Raedt, L., and Kimmig, A. 2015. Probabilistic (logic) programming concepts. *Machine Learning*.

De Raedt, L.; Frasconi, P.; Kersting, K.; and Muggleton, S. 2008. *Probabilistic Inductive Logic Programming — Theory and Applications*, volume 4911 of *Lecture Notes in Artificial Intelligence*. Springer.

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*.

Fierens, D.; Van den Broeck, G.; Renkens, J.; Shterionov, D.; Gutmann, B.; Thon, I.; Janssens, G.; and De Raedt, L. 2015. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming* 15(03):358–401.

Fuhr, N. 2000. Probabilistic Datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science (JASIS)* 51(2):95–110.

Getoor, L., and Taskar, B. 2007. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Goodman, N. D., and Stuhlmüller, A. 2014. The Design and Implementation of Probabilistic Programming Languages. `http://dippl.org`. Accessed: 2015-10-20.

Nilsson, U., and Maluszynski, J. 1995. *Logic, Programming, and PROLOG*. New York, NY, USA: John Wiley & Sons, Inc., 2nd edition.

Pfeffer, A. 2014. *Practical Probabilistic Programming*. Manning Publications.

Poole, D. 1993. Logic programming, abduction and probability. *New Generation Computing* 11:377–400.

Poole, D. 2000. Abducing through negation as failure: stable models within the independent choice logic. *Journal of Logic Programming* 44(1-3):5–35.

Poon, H., and Domingos, P. 2006. Sound and Efficient Inference with Probabilistic and Deterministic Dependencies. In *Proceedings of the 21st National Conference on Artificial Intelligence*, 458–463.

Renkens, J.; Kimmig, A.; Van den Broeck, G.; and De Raedt, L. 2014. Explanation-based approximate weighted model counting for probabilistic logics. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*.

Renkens, J.; Van den Broeck, G.; and Nijssen, S. 2012. k-optimal: A novel approximate inference algorithm for ProbLog. *Machine Learning* 89(3):215–231.

Riguzzi, F., and Swift, T. 2011. The PITA System: Tabling and Answer Subsumption for Reasoning under Uncertainty. *Theory and Practice of Logic Programming* 11(4–5):433–449.

Riguzzi, F. 2007. A Top Down Interpreter for LPAD and CP-Logic. In *10th Congress of the Italian Association for Artificial Intelligence (AI*IA)*.

Sato, T., and Kameya, Y. 2001. Parameter Learning of Logic Programs for Symbolic-Statistical Modeling. *J. Artif. Intell. Res. (JAIR)* 15:391–454.

Sato, T. 1995. A statistical learning method for logic programs with distribution semantics. In *Proceedings of*

the 12th International Conference on Logic Programming (ICLP).

Suciu, D.; Olteanu, D.; Christopher, R.; and Koch, C. 2011. *Probabilistic Databases*. Morgan & Claypool Publishers, 1st edition.

Van den Broeck, G.; Meert, W.; and Darwiche, A. 2014. Skolemization for weighted first-order model counting. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Vennekens, J.; Denecker, M.; and Bruynooghe, M. 2009. CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming* 9(3).

Vennekens, J.; Verbaeten, S.; and Bruynooghe, M. 2004. Logic programs with annotated disjunctions. In *In Proc. Int'l Conf. on Logic Programming*.

Vlasselaer, J.; Renkens, J.; Van den Broeck, G.; and De Raedt, L. 2014. Compiling probabilistic logic programs into sentential decision diagrams. In *Proceedings of the 1st Workshop on Probabilistic Logic Programming (PLP)*.

Vlasselaer, J.; Van den Broeck, G.; Kimmig, A.; Meert, W.; and De Raedt, L. 2015. Anytime inference in probabilistic logic programs with Tp-compilation. In *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*.